

Manual Técnico

1. INTRODUCCIÓN.....	5
1.1. DESCRIPCIÓN DEL SISTEMA.....	5
2. ANÁLISIS	5
2.1. ESPECIFICACIÓN DE REQUISITOS FUNCIONALES	5
2.2. ESPECIFICACIÓN DE REQUISITOS NO FUNCIONALES	5
2.3. MODELO DE DOMINIO	6
2.4. DESCRIPCIÓN DE CASOS DE USO	7
2.4.1. <i>Diagrama de Casos de Uso</i>	7
2.4.2. <i>Descripción detallada de Casos de Uso</i>	8
2.4.2.1 Iniciar partida	8
2.4.2.2 Iniciar carrera	8
2.4.2.3 Mover vehículo.....	8
2.4.2.4 Terminar carrera	9
2.4.2.5 Finalizar partida.....	9
2.4.2.6 Pausar carrera	10
2.4.2.7 Reanudar carrera.....	10
2.4.2.8 Salir de carrera	10
2.4.2.9 Salir del juego.....	10
2.5. DIAGRAMAS DE SECUENCIA.....	11
2.5.1. <i>Iniciar partida</i>	11
2.5.2. <i>Iniciar carrera</i>	12
2.5.3. <i>Mover vehículo</i>	12
2.5.4. <i>Terminar carrera</i>	13
2.5.5. <i>Finalizar partida</i>	13
2.5.6. <i>Pausar carrera</i>	13
2.5.7. <i>Reanudar carrera</i>	14
2.5.8. <i>Salir de carrera</i>	14
2.5.9. <i>Salir del juego</i>	14
3. DISEÑO	14
3.1. DIAGRAMA DE CLASES TOTAL.....	15
3.2. DIAGRAMAS DE CLASES PARCIALES.....	16
3.2.1. <i>AppDelegate</i>	16
3.2.2. <i>MainMenu</i>	17
3.2.3. <i>RaceConf</i>	17
3.2.4. <i>RankingMenu</i>	18
3.2.5. <i>Race</i>	19
3.2.6. <i>RaceMenu</i>	20
3.2.7. <i>EndRace</i>	20
3.3. DIAGRAMAS DE SECUENCIA DEL SISTEMA.....	21
3.3.1. <i>Iniciar partida</i>	21
3.3.2. <i>Iniciar carrera</i>	21
3.3.3. <i>Mover vehículo</i>	22
3.3.4. <i>Terminar Carrera</i>	22
3.3.5. <i>Finalizar partida</i>	23
3.3.6. <i>Pausar carrera</i>	23
3.3.7. <i>Reanudar carrera</i>	23
3.3.8. <i>Salir de carrera</i>	24
3.3.9. <i>Salir del juego</i>	24
4. DETALLES DE IMPLEMENTACIÓN.....	24
4.1. SPRITES.....	24
4.2. MAPA TMX	27

4.3. BUCLE PRINCIPAL	27
4.4. MÉTODOS CON EJECUCIÓN PROGRAMADA	27
4.5. COLISIONES	27
4.6. EVENTOS.....	27

1. Introducción

1.1. Descripción del sistema

SuperCars es un videojuego de carreras en 2D. En él, el jugador debe mover el vehículo a izquierda y derecha para evitar los obstáculos que van apareciendo en la carretera, así como adelantar a los oponentes. Se pretende que la aleatoriedad en la generación de obstáculos y oponentes provoque situaciones distintas en cada partida, así como el incremento de dificultad.

2. Análisis

2.1. Especificación de Requisitos Funcionales

En este apartado se reflejan todos los Requisitos Funcionales que se han establecido al inicio del proyecto o bien se han añadido posteriormente. Éstos reflejan toda la funcionalidad que es imprescindible para considerar el proyecto como finalizado.

La nomenclatura es RF (Requisito Funcional), seguido de un número de identificación único. Ésta nomenclatura puede ser útil para relacionar cualquier parte de la documentación con los requisitos.

# Requisito	Descripción
RF01	Deben poder configurarse varios parámetros de la carrera como vueltas, rivales y dificultad
RF02	El jugador debe poder controlar su vehículo con unos botones que se muestren en pantalla
RF03	La vista del juego debe ser aérea (en 2D)
RF04	Debe haber un <i>HUD</i> ¹ que muestre al jugador información sobre la carrera, como vueltas o posición
RF05	Debe guardarse un <i>ranking</i> de vueltas rápidas
RF06	La dificultad debe influir en el comportamiento de los rivales
RF07	El jugador debe poder pausar la carrera en cualquier momento

2.2. Especificación de Requisitos No Funcionales

Aquí se detallan los Requisitos No Funcionales. Éstos reflejan características del sistema que, si bien su incumplimiento no hace que el sistema deje de funcionar, sí pueden influir en la experiencia del usuario al utilizar la aplicación.

¹ HUD son las siglas en inglés de Head-Up Display. Es una pantalla transparente que presenta la información al usuario sin que éste tenga que cambiar su punto de vista para verla. El origen proviene de que el usuario pueda ver la información con la cabeza erguida (Head-Up) y mirando al frente, sin necesidad de bajarla.

La nomenclatura sigue el mismo patrón que en los *Requisitos Funcionales* (2.1), siendo RNF (Requisitos No Funcionales) seguido de un número de identificación único, además de poseer la misma utilidad.

# Requisito	Descripción
RNF01	La aplicación debe mantener una tasa de <i>frames per second</i> superior a 30
RNF02	La tasa de <i>frames per second</i> debe ser estable (no variar más de un 25% cada segundo)
RNF03	La carrera debe poder contener 40 o más obstáculos simultáneos
RNF04	La aplicación debe funcionar en modo <i>portrait</i> ² para ofrecer suficiente visibilidad al jugador
RNF05	Al iniciar la aplicación, el menú principal debe ser visible en menos de 5 segundos
RNF06	La transición entre escenas no debe durar más de 2 segundos

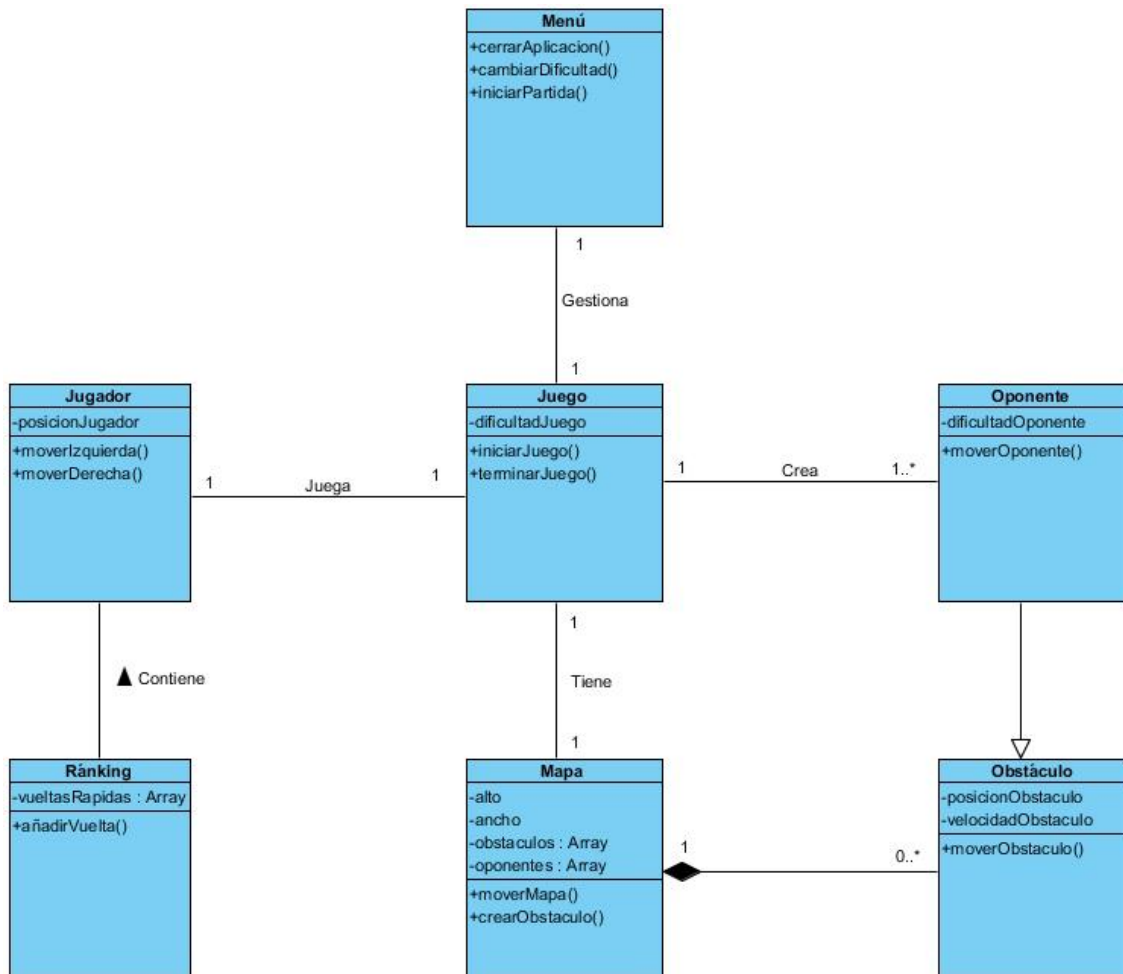
2.3. Modelo de Dominio

En el Modelo de Dominio se reflejan las clases (conceptuales) que se consideran significativas para el dominio del problema.

Estas clases no tienen por qué coincidir con las implementadas finalmente, ya que éstas últimas pueden variar en función del entorno de desarrollo e incluso del de despliegue. La lógica y funcionalidades descritas en estas clases sí deben mantenerse.

La versión final del Modelo de Dominio que refleja el diseño real de la aplicación se puede encontrar en el apartado 3.1.

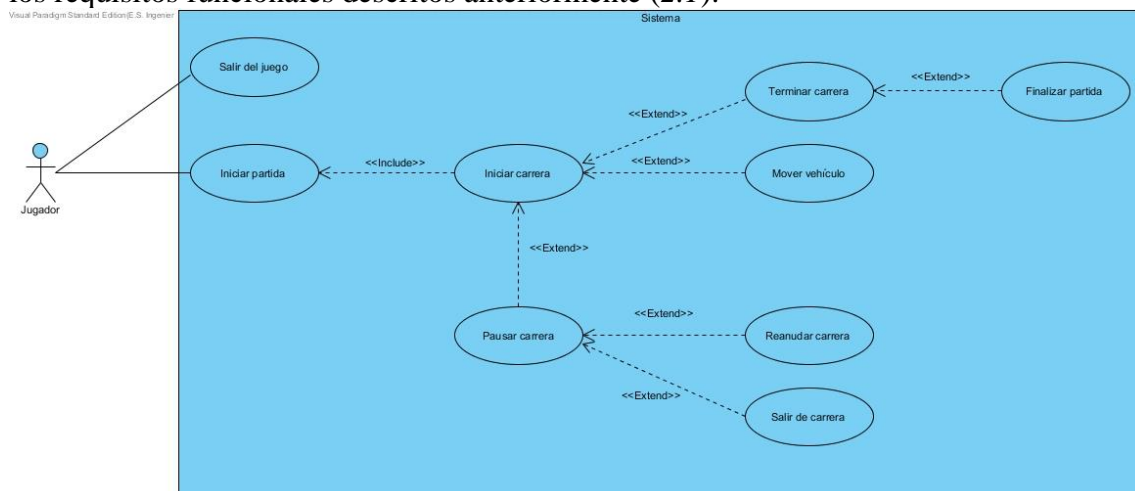
² *Portrait* hace referencia a la orientación de la aplicación, que debe extender su lado más largo de arriba a abajo de la pantalla (orientación vertical)



2.4. Descripción de Casos de Uso

2.4.1. Diagrama de Casos de Uso

Este diagrama representa las interacciones del usuario con el sistema, cumpliendo con los requisitos funcionales descritos anteriormente (2.1).



A continuación se realiza una descripción detallada de los Casos de Uso, que incluye:

- Breve descripción.

- Precondiciones.
- Postcondiciones.
- Flujo de eventos.
- Flujo alternativo (opcional).

Se obvia el actor debido a que en todos ellos es el jugador.

2.4.2. Descripción detallada de Casos de Uso

2.4.2.1 Iniciar partida

Iniciar partida			
Descripción breve	El jugador pulsa el botón <i>Play</i> en <i>MainMenu</i> y la escena cambia a <i>RaceConf</i>		
Precondiciones	La escena activa es <i>MainMenu</i>		
Postcondiciones	La escena activa es <i>RaceConf</i>		
Flujo Principal	Jugador		Sistema
	1	Pulsa el botón <i>Play</i>	
	2		Cambia escena activa de <i>MainMenu</i> a <i>RaceConf</i>

2.4.2.2 Iniciar carrera

Iniciar carrera			
Descripción breve	El jugador configura las opciones de carrera y ésta da comienzo de acuerdo con ellas		
Precondiciones	La escena activa es <i>RaceConf</i>		
Postcondiciones	La escena activa es <i>Race</i>		
Flujo Principal	Jugador		Sistema
	1		Muestra parámetros de configuración de la carrera
	2	Ajusta los parámetros de configuración	
	3		Recoge la configuración introducida e inicia la carrera de acuerdo a dicha configuración

2.4.2.3 Mover vehículo

Mover vehículo				
Descripción breve		El jugador pulsa un botón de dirección y el vehículo se mueve al sitio correspondiente		
Precondiciones		La escena activa es <i>Race</i>		
Postcondiciones		El vehículo se ha movido a la posición deseada		
Flujo Principal		Jugador		Sistema
		1	Pulsa botón de	

		movimiento <i>leftArrow</i> o <i>rightArrow</i>	
	2		Identifica cuál de los botones ha sido pulsado
	3		Mueve el vehículo al lado correspondiente
	4		Se queda a la espera de una nueva interacción
Flujo Alternativo [A3]	Jugador		Sistema
	1		Si está a la izquierda y se ha pulsado <i>leftArrow</i> , no hace nada. Vuelve al paso 4
Flujo alternativo [A3]	Jugador		Sistema
	1		Si está a la derecha y se ha pulsado <i>rightArrow</i> , no hace nada. Vuelve al paso 4

2.4.2.4 Terminar carrera

Terminar carrera			
Descripción breve	El jugador alcanza la línea de meta de la última vuelta y la carrera finaliza		
Precondiciones	La escena activa es <i>Race</i>		
Postcondiciones	La escena activa es <i>EndRace</i>		
Flujo Principal	Jugador		Sistema
	1	Alcanza la línea de meta	
	2		Finaliza la carrera
	3		Cambia escena activa de <i>Race</i> a <i>EndRace</i>

2.4.2.5 Finalizar partida

Finalizar partida			
Descripción breve	El jugador vuelve a <i>MainMenu</i> pulsando <i>back</i> en la escena <i>EndRace</i> , una vez terminada la carrera		
Precondiciones	La escena activa es <i>EndRace</i>		
Postcondiciones	La escena activa es <i>MainMenu</i>		
Flujo Principal	Jugador		Sistema
	1	Pulsa <i>back</i>	
	2		Cambia escena activa de <i>EndRace</i> a <i>MainMenu</i>

2.4.2.6 Pausar carrera

Pausar carrera			
Descripción breve	El jugador pulsa el botón <i>menú</i> en la escena <i>Race</i> y se pausa el juego para mostrar <i>RaceMenu</i>		
Precondiciones	La escena activa es <i>Race</i>		
Postcondiciones	La escena activa es <i>RaceMenu</i>		
Flujo Principal	Jugador		Sistema
	1	Pulsa botón <i>menu</i>	
	2		Pausa escena <i>Race</i>
	3		Muestra escena <i>RaceMenu</i>

2.4.2.7 Reanudar carrera

Reanudar carrera					
Descripción breve		El jugador pulsa <i>resume</i> en <i>RaceMenu</i> y se reanuda la carrera			
Precondiciones		La escena activa es <i>RaceMenu</i>			
Postcondiciones		La escena activa es <i>Race</i>			
Flujo Principal		Jugador		Sistema	
		1	Pulsa <i>resume</i>		
		3		Cambia escena activa de <i>RaceMenu</i> a <i>Race</i>	

2.4.2.8 Salir de carrera

Salir de carrera			
Descripción breve	El jugador pulsa el botón <i>Exit</i> en <i>RaceMenu</i> y la escena cambia a <i>MainMenu</i>		
Precondiciones	La escena activa es <i>RaceMenu</i>		
Postcondiciones	La escena activa es <i>MainMenu</i>		
Flujo Principal	Jugador		Sistema
	1	Pulsa el botón <i>exit</i>	
	2		Finaliza la escena <i>Race</i>
	3		Cambia escena activa de <i>RaceMenu</i> a <i>MainMenu</i>

2.4.2.9 Salir del juego

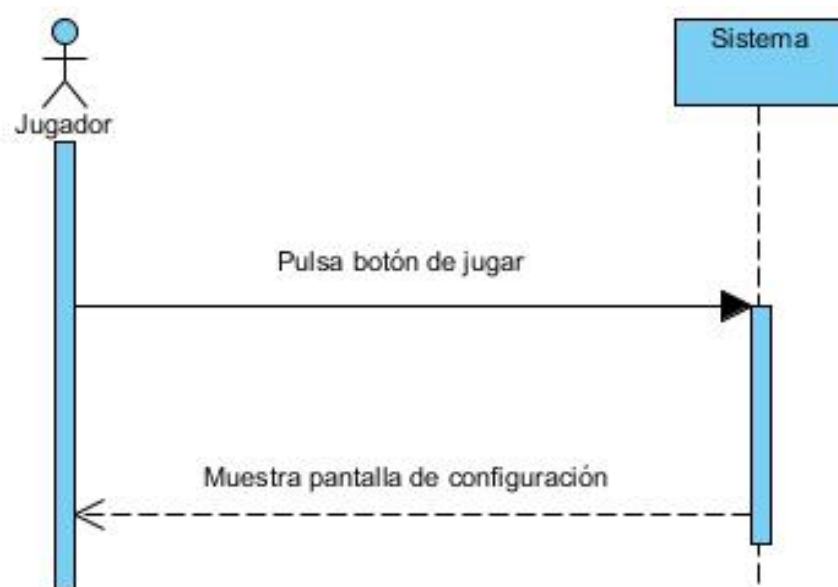
Salir del juego					
Descripción breve		El jugador pulsa el botón <i>exit</i> de <i>MainMenu</i> y la aplicación se cierra			
Precondiciones		La escena activa es <i>MainMenu</i>			
Postcondiciones		Aplicación cerrada correctamente			
Flujo Principal		Jugador		Sistema	
		1	Pulsa <i>exit</i> en <i>MainMenu</i>		

2.5. Diagramas de Secuencia

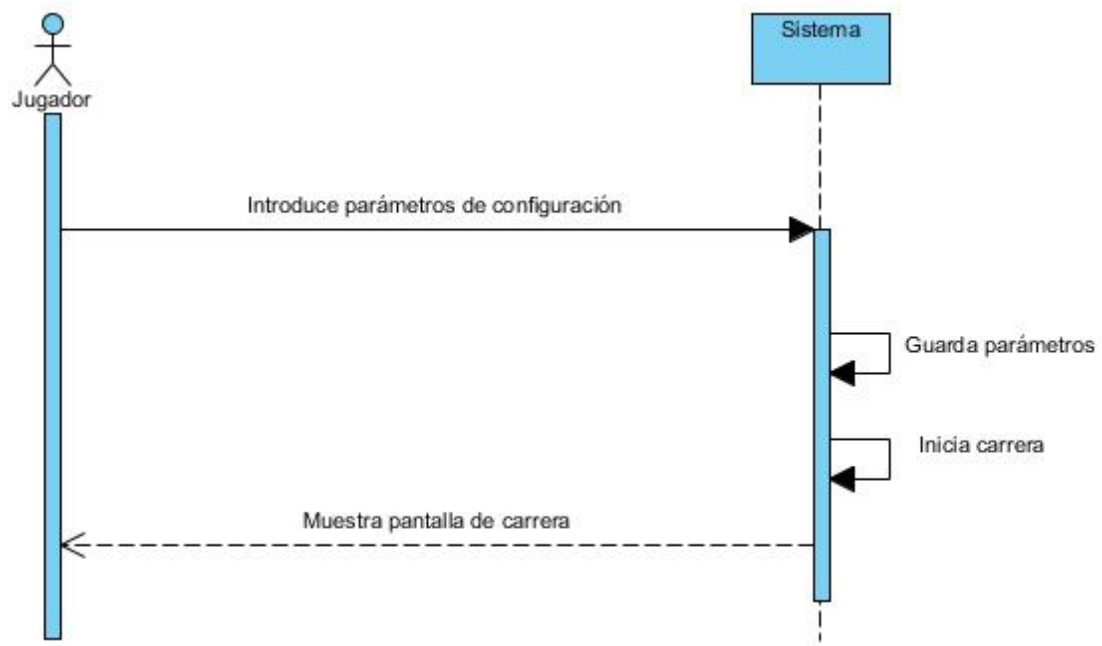
Aquí se muestran los diagramas de secuencia que reflejan el flujo de información que se produce entre el usuario y el sistema, según las clases y casos de uso de análisis.

Ésto es ampliado con los *Diagramas de Secuencia del Sistema* (3.3) que reflejan también el flujo interno del sistema.

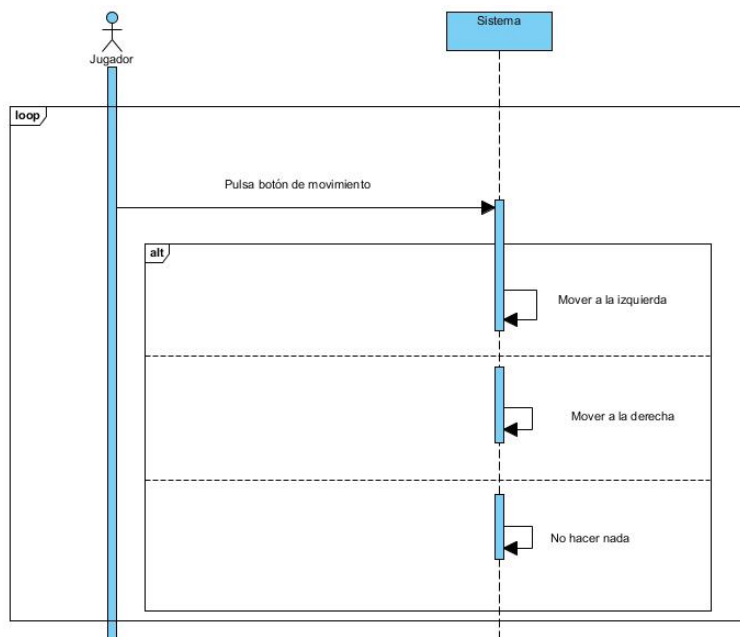
2.5.1. Iniciar partida



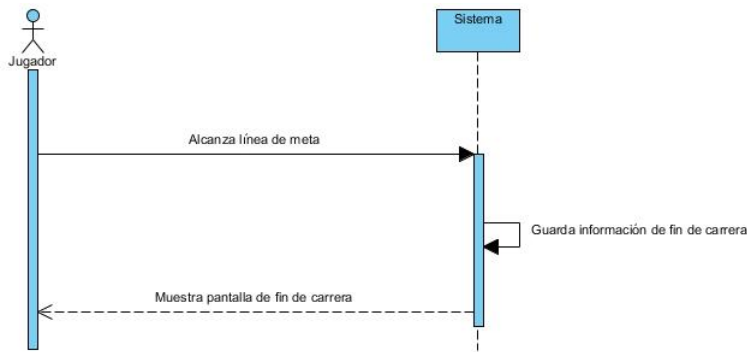
2.5.2. Iniciar carrera



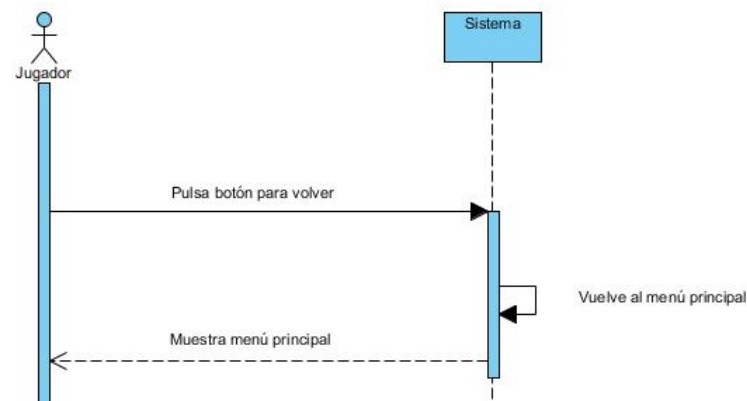
2.5.3. Mover vehículo



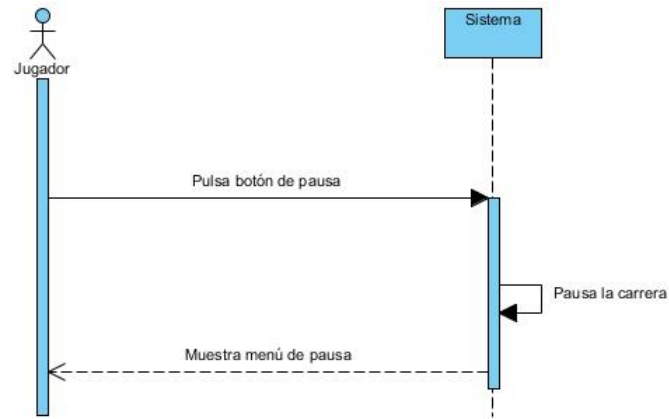
2.5.4. Terminar carrera



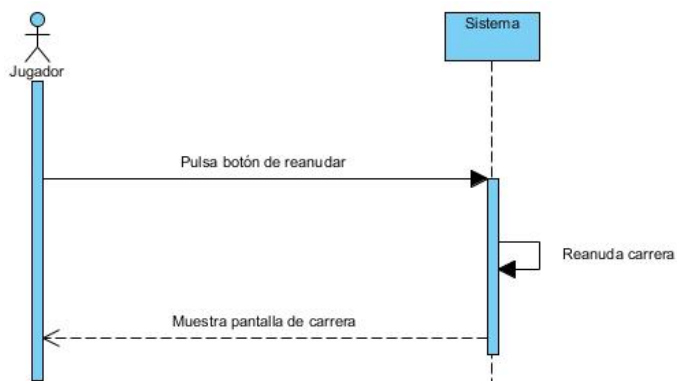
2.5.5. Finalizar partida



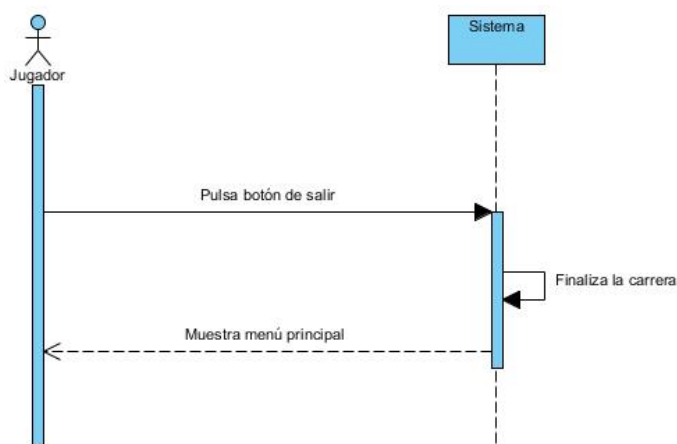
2.5.6. Pausar carrera



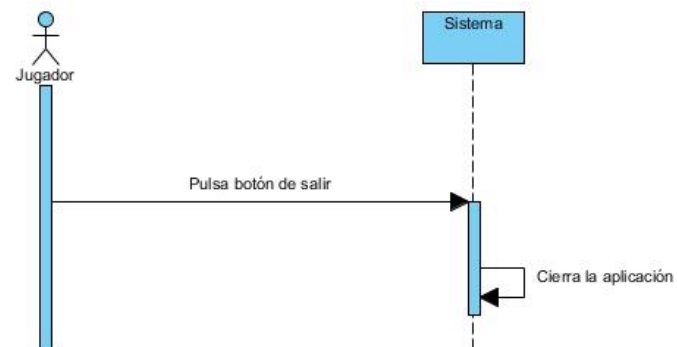
2.5.7. Reanudar carrera



2.5.8. Salir de carrera



2.5.9. Salir del juego



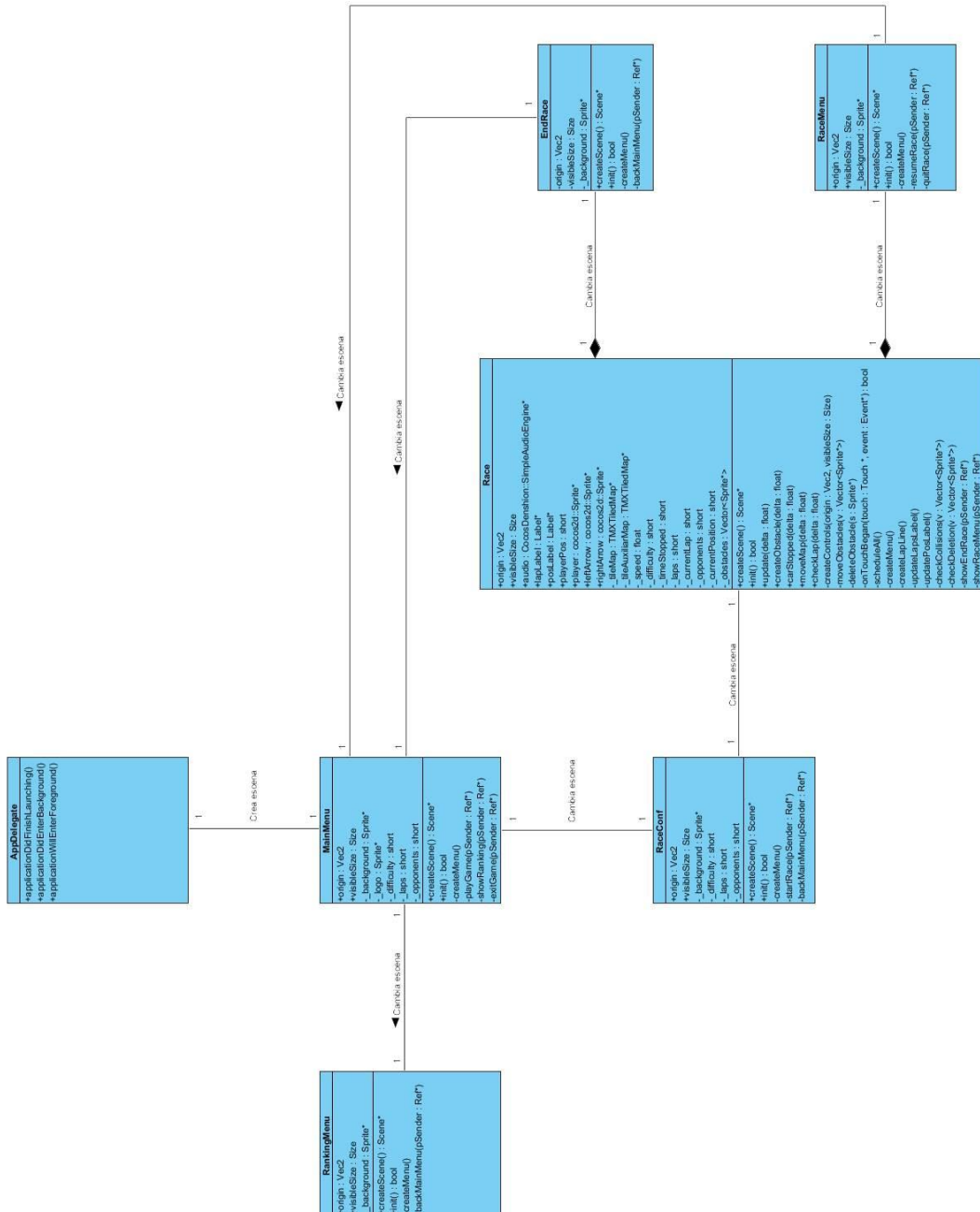
3. Diseño

Las variaciones en el diseño con respecto al *Análisis (2)* están producidas por la adecuación del desarrollo al *framework* Cocos2d-x.

A continuación se muestra el diagrama de clases total, seguido de un diagrama parcial de cada clase.

3.1. Diagrama de clases total

Este es el diagrama de clases total que finalmente refleja el diseño de la aplicación. Es diferente del Modelo de Dominio (2.3) de la fase de *Análisis* pero mantiene la funcionalidad descrita.



De las diferencias más importantes entre éste diagrama de clases final y el *Modelo de Dominio* (2.3) de la fase de *Análisis*, cabe destacar:

- La entidad Jugador no es necesario representarla como una clase separada, ya que finalmente es una variable de la escena *Race* (3.2.5) y la clase *Sprite* propia de Cocos2d-x es suficiente para implementar la funcionalidad deseada. Sucede

lo mismo con las clases *Mapa*, *Obstáculo* y *Oponente*. En el caso de éstos últimos incluso son tratados en un mismo vector, diferenciados por un *Tag*³

3.2. Diagramas de clases parciales

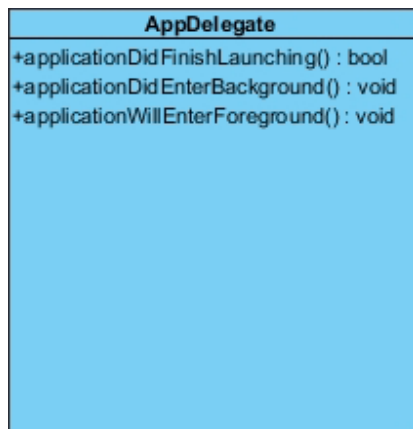
En este apartado se detallan las clases parciales del sistema, que se pueden identificar con una escena del juego (exceptuando *AppDelegate*).

Todas las clases tienen su diagrama parcial, una breve explicación y sus métodos y variables más importantes.

Existen ciertas variables y métodos que son comunes a casi todas las clases, por lo que se detallan aquí:

- *origin*: variable de tipo *Vec2* que contiene las coordenadas X e Y del origen.
- *visibleSize*: variable de tipo *Size* que contiene la altura y el ancho de la pantalla.
- *createScene()*: es el constructor de la escena, por lo que devuelve un objeto escena básico.
- *init()*: función que se ejecuta siempre y contiene las llamadas principales de la clase. Se puede considerar como una función *main*⁴, por lo que el resto de métodos son llamados (directa o indirectamente) desde éste.
- *createMenu()*: crea y coloca los botones del menú correspondiente a esa escena.
- *onTouchBegan(Touch* touch, Event* event)*: se asocia al listener de los botones para que se ejecute cuando el jugador los toca.

3.2.1. AppDelegate



Esta es una clase base propia de Cocos2d-x. Su función es la de gestionar el inicio y fin de la aplicación.

- *applicationDidFinishLaunching()*: se ejecuta cuando la aplicación termina de abrirse.
- *applicationDidEnterBackground()*: se ejecuta cuando la aplicación pasa a segundo plano.

³ Cocos2d-x utiliza *Tags* en sus clases como un identificador que puede ser único o compartido por distintos elementos.

⁴ Función típicamente llamada al arrancar un programa

- *applicationWillEnterForeground()*: se ejecuta cuando la aplicación vuelve activarse tras *applicationDidEnterBackground()*.

3.2.2. MainMenu

MainMenu
+origin : Vec2 +visibleSize : Size -_background : Sprite* -_logo : Sprite*
+createScene() : Scene* +init() : bool -createMenu() : void -playGame(pSender : Ref*) : void -showRanking(pSender : Ref*) : void -exitGame(pSender : Ref*) : void

Esta clase se corresponde con el menú principal del juego. Es por tanto la escena creada desde

AppDelegate.

- *playGame(Ref* pSender)*: crea una nueva instancia de RaceConf y reemplaza a ésta.
- *showRanking(Ref* pSender)*: crea una nueva instancia de RankingMenu y reemplaza a ésta.
- *exitGame(Ref* pSender)*: finaliza la aplicación.

3.2.3. RaceConf

RaceConf
+origin : Vec2 +visibleSize : Size -_background : Sprite* -_difficulty : short -_laps : short -_opponents : short
+createScene() : Scene* +init() : bool -createMenu() : void -createConfMenu() : void -startRace(pSender : Ref*) : void -backMainMenu(pSender : Ref*) : void -diffUpdate(delta : float) : void -oppUpdate(delta : float) : void -lapsUpdate(delta : float) : void

Esta clase corresponde a la escena de configuración de la carrera.

- *startRace(Ref* pSender)*: recoge y guarda los parámetros introducidos, después crea una nueva instancia de RaceScene y reemplaza a ésta.

- *backMainMenu(Ref* pSender)*: vuelve a *MainMenu*.

3.2.4. RankingMenu

RankingMenu
+origin : Vec2 +visibleSize : Size - _background : Sprite*
+createScene() : Scene* +init() : bool -createMenu() : void -backMainMenu(pSender : Ref*) : void

Esta clase corresponde a la escena de visualización del *ranking* de vueltas rápidas.

- *backMainMenu(Ref* pSender)*: vuelve a *MainMenu*.

3.2.5. Race

Race
<pre> +origin : Vec2 +visibleSize : Size +audio : CocosDenshion::SimpleAudioEngine* - _carFiles : std::vector<std::string> - _tileMap : TMXTiledMap* - _tileAuxiliarMap : TMXTiledMap* - _player : cocos2d::Sprite* - _leftArrow : cocos2d::Sprite* - _rightArrow : cocos2d::Sprite* - _obstacles : Vector<Sprite*> - _opponents : Vector<Sprite*> - _lapsTime : float[] - _lapLabel : Label* - _posLabel : Label* - _timerLabel : Label* - _speed : short - _checkOppIsOut : short - _difficulty : short - _timeStopped : short - _laps : short - _currentLap : short - _numOpponents : short - _currentPosition : short - _fastestLap : float - _playerPos : short - _time : float +createScene() : Scene* +init() : bool -update(delta : float) : void -createObstacle(delta : float) : void -carStopped(delta : float) : void -moveMap(delta : float) : void -moveObstacles(delta : float) : void -moveOpponents(delta : float) : void -stoppedOpponents(delta : float) : void -moveCrashedOpponents(delta : float) : void -moveInvOpponents(delta : float) : void +checkPosition(delta : float) : void -avoidCollision(delta : float) : void -checkOppIsOut(delta : float) : void -checkLap(delta : float) : void -timerMethod(delta : float) : void -spawnOpponents() : void -getRandomSpawnX(min : short, max : short) : short -createControls(origin : Vec2, visibleSize : Size) : void -deleteObstacle(s : Sprite*) : void -onTouchBegan(touch : Touch *, event : Event*) : bool -scheduleAll() : void -createMenu() : void -createLapLine() : void -updateLapsLabel() : void -updatePosLabel() : void -changeXOpponent(s : Sprite*) : void -checkPlayerCollisions(v : Vector<Sprite*>) : void -checkOpponentCollisions(v : Sprite*) : void -checkDeletion(v : Vector<Sprite*>) : void -showEndRace(pSender : Ref*) : void -showRaceMenu(pSender : Ref*) : void </pre>

Esta clase es la escena principal del juego. Contiene todo lo visible en la escena de carrera.

- `_carFiles`: es un vector de *strings* que guardan el nombre de los archivos de textura de los vehículos.
- `_difficulty`, `_laps` y `_numOpponents`: estas variables se recogen de la información introducida en *RaceConf*.
- `_speed`: esta variable controla cómo de rápido se mueve el mapa y los rivales. Se calcula a partir de `_difficulty` y `_opponents`.

- *scheduleAll()*: este método programa (*schedule*) los métodos que deben ejecutarse cada *frame* o cada tiempo estimado.
- *update(float delta)*: bucle principal del juego. Se ejecuta cada *frame* y *delta* es el tiempo transcurrido entre el *frame* actual y el anterior (este tiempo es variable). Desde este método se realizan llamadas a *updatePosLabel()*, *moveObstacles(_obstacles)*, *checkCollisions(_obstacles)* y *checkDeletion(_obstacles)*.
- *carStopped(float delta)*: éste método pasa a ser el bucle principal cuando el vehículo del jugador se para (ha colisionado con un obstáculo). Tras un tiempo llama de nuevo a *scheduleAll()* y se desactiva.
- *moveMap(float delta)*: mueve el mapa en función de *_speed*.
- *moveObstacles(Vector<Sprite*> v)*: mueve los obstáculos en función de *_speed* para generar la sensación de que éstos están fijos en el mapa.
- *createObstacle(float delta)*: crea un obstáculo cada *delta* segundos en una posición semi-aleatoria.
- *checkLap(float delta)*: comprueba cada *delta* segundos si la vuelta actual es igual al número máximo de vueltas, para terminar la carrera.

3.2.6. RaceMenu

RaceMenu
+origin : Vec2 +visibleSize : Size - _background : Sprite*
+createScene() : Scene* +init() : bool -createMenu() : void -resumeRace(pSender : Ref*) : void -quitRace(pSender : Ref*) : void

Esta escena es el menú llamado desde *Race*.

- *resumeRace(Ref* pSender)*: reanuda la carrera.
- *quitRace(Ref* pSender)*: finaliza la carrera, crea una nueva instancia de *MainMenu* y se reemplaza por ésta.

3.2.7. EndRace

EndRace
-origin : Vec2 -visibleSize : Size - _background : Sprite*
+createScene() : Scene* +init() : bool -createMenu() : void -backMainMenu(pSender : Ref*) : void

Esta es la escena que sigue a *Race* cuando la carrera termina. Muestra el tiempo obtenido y permite volver a *MainMenu*.

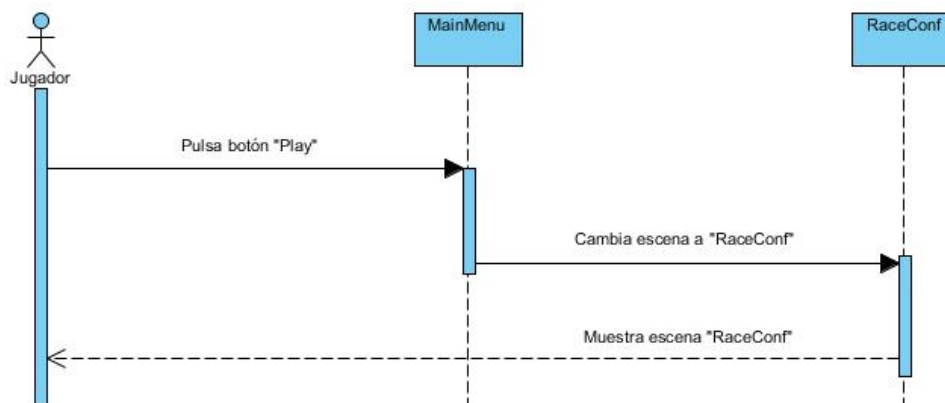
- *backMainMenu(Ref* pSender)*: devuelve a *MainMenu*.

3.3. Diagramas de Secuencia del Sistema

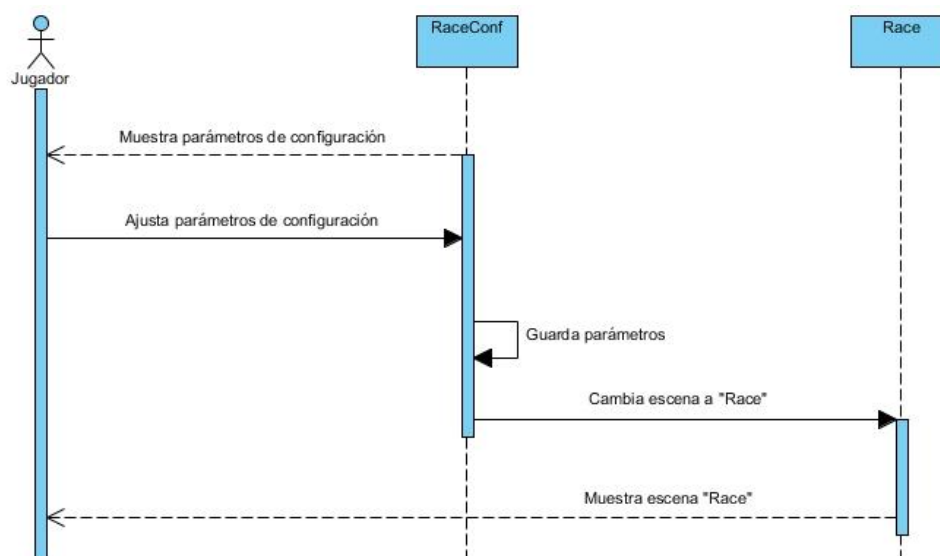
En estos diagramas se pueden ver las transiciones de información y los cambios que se producen en el sistema debido a la interacción del Jugador.

Por motivos de legibilidad se ha decidido describir las relaciones con frases cortas en lugar de indicar el nombre del método correspondiente, aunque cada relación entre clases del sistema corresponde con la llamada a un método.

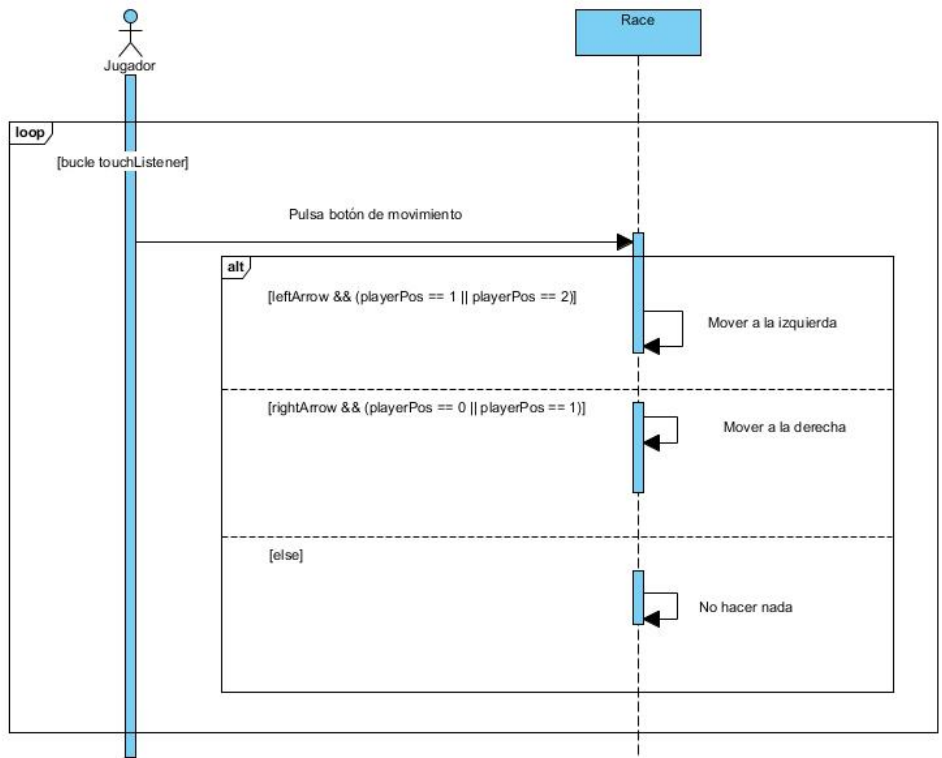
3.3.1. Iniciar partida



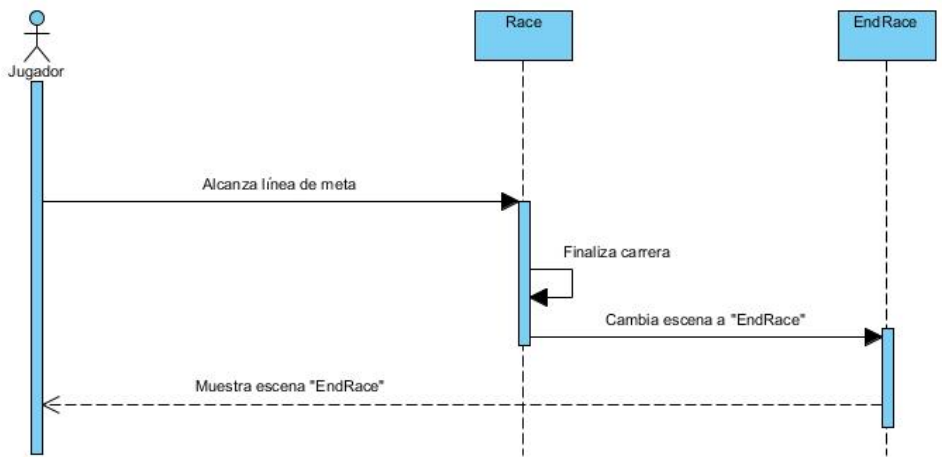
3.3.2. Iniciar carrera



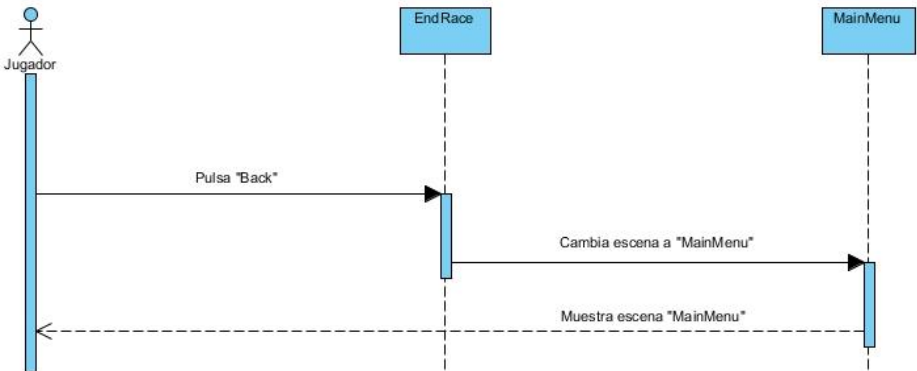
3.3.3. Mover vehículo



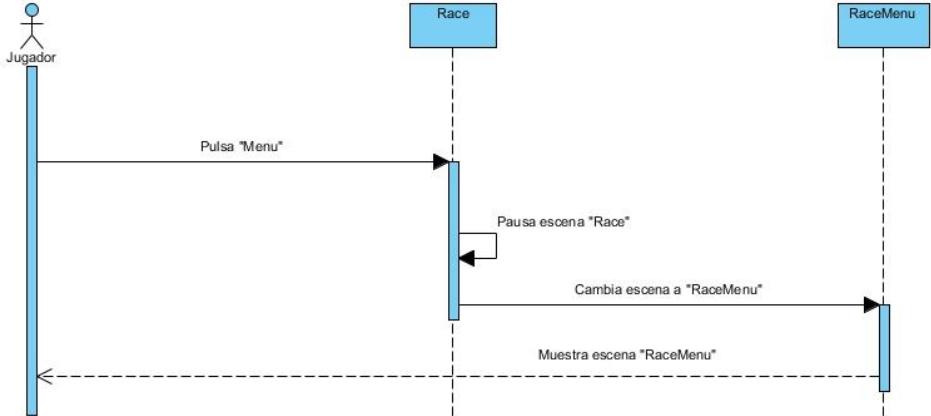
3.3.4. Terminar Carrera



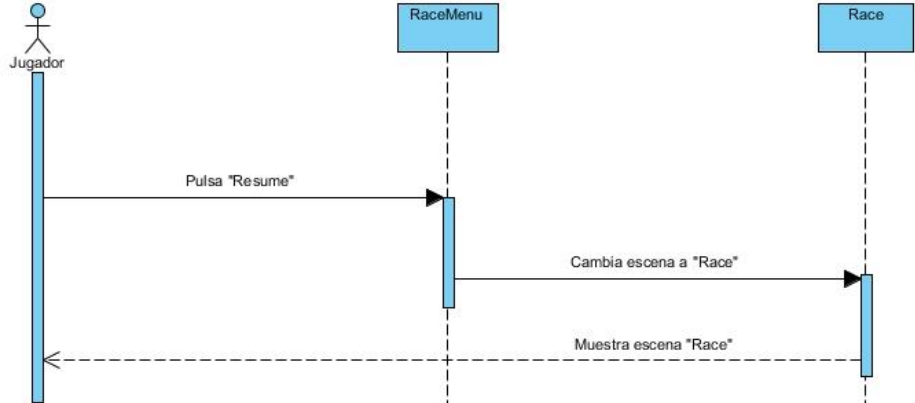
3.3.5. Finalizar partida



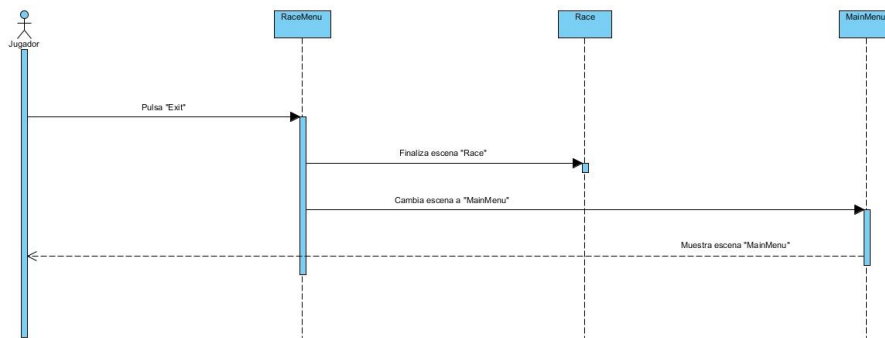
3.3.6. Pausar carrera



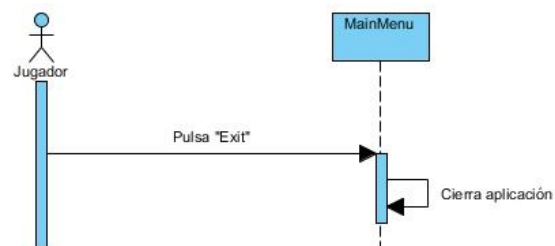
3.3.7. Reanudar carrera



3.3.8. Salir de carrera



3.3.9. Salir del juego



4. Detalles de implementación

Aquí se explican los detalles de implementación que se considera que son importantes, bien por ser características muy específicas de Cocos2d-x o del tipo de proyecto.

4.1. Sprites

El jugador, los rivales, los obstáculos... Todos ellos son *Sprites*. Un *Sprite*, en 2D, es una imagen que se manipula modificando sus propiedades de posición, rotación, escala u opacidad, entre otras. Por lo tanto, cualquier imagen visible en la pantalla puede ser un *Sprite*.

Para crear un *Sprite* en Cocos2d-x, tan sólo se necesita 1 línea:

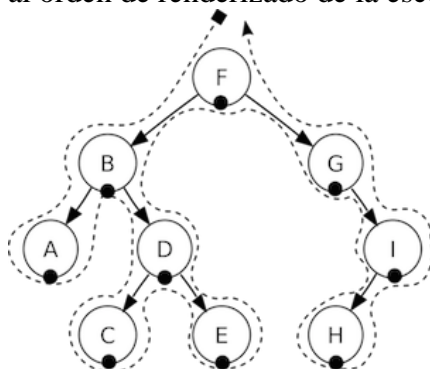
```
_player = Sprite::create("audi_r8.png");
```

Tras crearlo, hay que añadirlo al árbol de nodos de la escena, de la siguiente forma:

```
this->addChild(_player, 100);
```

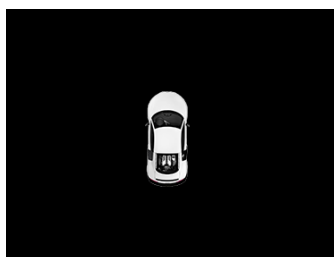
Donde, el segundo parámetro, es el orden de prioridad (*Z-order*) de la propia escena. Es un parámetro opcional, pero es recomendable gestionarlo ya que, habiendo dos *Sprites*

en la misma posición, el que mayor *Z-order* tiene se muestra por encima. Esto se debe al orden de renderizado de la escena, que se puede ver en el *scene graph*⁵:



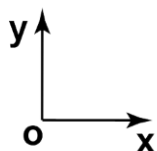
El *Renderer*⁶ recorre el árbol empezando por la izquierda y hacia abajo, después la raíz y después la parte derecha, de forma recursiva. Así, si suponemos que *B* y *G* son dos elementos en la misma posición, *G* se superpondría a *B*, ya que se renderiza después. Esto también es aplicable a otro tipo de nodos, que también son añadidos al árbol de la escena. Por defecto, al añadir un *Sprite*, su *Z-order* es superior al de los elementos añadidos anteriormente.

Se presupone que, en el directorio *Resources/* del proyecto, existe un archivo *.png* llamado *audi_r8.png*. Lo que se obtendría, sería algo como esto:



Nótese que el fondo, en este caso, tan sólo refleja la ausencia de otros elementos, siendo negro el color por defecto. En el caso del *Sprite* que se ha añadido, tiene transparencia ya que es un *.png*, algo interesante para visualizar formas que no sean rectangulares.

A continuación, es habitual configurar el *AnchorPoint*. Esto es, un punto (dentro del *Sprite*) sobre el que se aplican las transformaciones y cuyo valor por defecto es (0.5, 0.5). Este valor por defecto se correspondería con el centro, siendo 1 el valor máximo en ambas coordenadas y 0 el mínimo. El eje de coordenadas que utiliza Cocos2d-x es el mismo que utiliza OpenGL, el cual se rige por la “Regla de la Mano Derecha” (Right Handed Cartesian Coordinate System).



Por lo tanto, el *AnchorPoint* por defecto sería el siguiente:

⁵ El *scene graph* es representado como un árbol y contiene todos los nodos pertenecientes a la escena

⁶ Sistema de Cocos2d-x que se encarga de renderizar la escena



Para cambiarlo, tan sólo hay que pasar como parámetro un *Vec2*, un tipo de dato de Cocos2d que representa una coordenada en 2D. Se haría de esta forma:

```
_player->setAnchorPoint(Vec2(0.5, 1));
```

Así, habremos modificado el *AnchorPoint* en su eje Y:



Es importante prestar atención al *AnchorPoint* de un *Sprite* porque, sino, es probable que éste sea mal posicionado en algún momento.

La modificación de otras propiedades como *scale*, *position*, *skew* o *rotation*, se realiza de forma análoga, teniendo siempre en cuenta el *AnchorPoint*.

Habitualmente, los *Sprites* se utilizan para representar objetos que deben moverse por la pantalla. Para esto, se pueden crear *Actions*, que después uno o varios *Sprites* ejecutan. En todos los juegos se utilizan *Actions* de alguna forma, ya que es lo que permite modificar los *Sprites* a lo largo del tiempo. Algunas de las más habituales son: *MoveBy* y *MoveTo*. La diferencia entre estas dos reside en que, mientras *MoveBy* indica en qué dirección mover el nodo a lo largo de un tiempo *t*, con *MoveTo* se indica a qué posición concreta debe moverse el nodo a lo largo de ese tiempo.

Para crear una acción, es necesaria una sentencia como la siguiente:

```
auto moveRight = MoveBy::create(0.25, Vec2(100, 0));
```

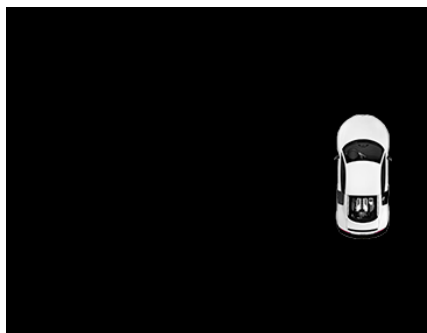
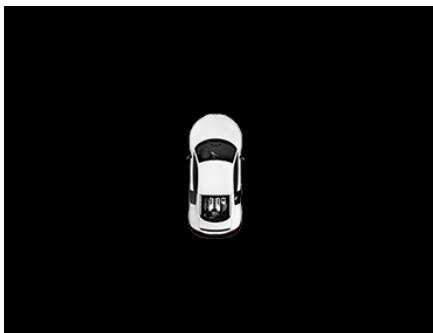
Con la palabra reservada *auto*, se evita tener que especificar el tipo de la variable. En este caso, se indica que, el *Sprite* que ejecute esta acción, debe moverse 100px⁷ en el eje X, durante 0.25s. Si el *Sprite* está en la posición (50, 50), tendría que aumentar su coordenada X en 100 (hasta llegar a 150), a lo largo de 0.25s.

Para que el *Sprite* que se creó antes, *_player*, ejecute esta *Action*, tan sólo hay que añadir:

```
_player->runAction(moveRight);
```

⁷ px es la nomenclatura habitual para "Pixel"

Por lo tanto, suponiendo que esto se añade inmediatamente después de la creación del *Sprite*, obtendríamos el siguiente resultado:



Nótese que la proporción de píxeles no se ha tenido en cuenta.

4.2. Mapa TMX

4.3. Bucle principal

4.4. Métodos con ejecución programada

4.5. Colisiones

4.6. Eventos